allenCahn tutorial

Updated 4th November, 2014

1 Parameters

This section describes the model parameters that the user can access through the `parameters.h` file in each application folder. These can be divided between model-specific parameters and those generic to all models, and there are three categories of model-specific parameters: constant coefficients and tensors, bulk free energy functions, and residuals. In the context of their model, constant coefficients are self-explanatory. Bulk free energy functions and their derivatives are incorporated as functions of the phase field variables, usually instantiated in the code as `n` or `c`. All of the examples and defaults use polynomial free energies, and their derivatives must be supplied by the user. Non-polynomial functions could be implemented, but are not supported at this time. Residuals are used directly in evolution. Changing them is not recommended unless the user is comfortable with deal.ii and the PRISMS framework.

1.1 Generic Parameters

- `problemDIM`
  Dimension of the problem (e.g. 1D, 2D, 3D)

- `spanX`
  Length of system in x-direction

- `spanY`
  Length of system in y-direction. Not used if `problemDIM` < 2

- `spanZ`
  Length of system in z-direction. Not used if `problemDIM` < 3

- `refineFactor`
  Defines the refinement of the mesh. There are $2^{\text{refineFactor}}$ elements in each direction in this implementation, and $\left(2^{\text{refineFactor}}\right)^{\text{problemDIM}}$ elements in total.

- `finiteElementDegree`
  The order of interpolation of the finite element space. In this case, the order of the Lagrange elements to be used.

- `dt`
  The simulation timestep.

- `numIncrements`
  The number of simulation time steps. Final time is then `dt`·`numIncrements`.

- `writeOutput`
  Whether we are writing any output. Takes a boolean argument, e.g. `true`.

- `skipOutputSteps`
  Output will be written every `skipOutputSteps` iterations. If `writeOutput` `true`, the initial conditions will always be written.

1.2 `allenCahn` Parameters

Given the evolution equation:

$$\frac{\partial \eta}{\partial t} = -M_\eta \left(f'(\eta) - \kappa_\eta^2 \nabla^2 \eta \right) \tag{1}$$

- `Mn`
  Allen-Cahn mobility parameter M_η in Eq. (1).

- `Kn`
  Square of the Allen-Cahn gradient energy parameter, κ_η^2 in Eq. 1. (tensor form ????)

- `fnV`
  First derivative of the Allen-Cahn bulk free energy $f(\eta)$, where η is represented in code by `n`. The default for $f(\eta)$ in the code is the double well potential:

$$f(\eta) = \eta^2(1-\eta)^2 \tag{2}$$

$$f'(\eta) = 4\eta(\eta-1)(\eta-1/2)$$

- Residuals
  In the context of Eq. 1, we have

$$\texttt{rnV} = \eta - \Delta t \cdot M_\eta f'(\eta)$$

$$\texttt{rnxV} = \nabla \eta$$

  where Δt is the simulation timestep.

2 Initial Conditions

Initial conditions for the phase field parameters are generated in the functions `InitialConditionN` or `InitialConditionC` located in the `main.cc` file in the application folder. Two types of initial conditions have been implemented so far: a field of random noise around some mean value, and analytical functions of position. Random noise is the default for at least one variable in all of the phase field codes. Here, as an example, c is initialized to random values in the range 0.019-0.021:

```
template <int dim>
double InitialConditionC<dim>::value (const Point<dim> &p, const unsigned int
    /* component */) const
{
        //set result equal to the concentration initial condition
        return 0.02 + 1.0e-3*(2*(0.5 - (double)(std::rand() % 100 )/100.0));
}
```

Analytical functions of position can be generated from the members of the class `Point`, e.g. `p.distance` or the x and y coordinates, `p[0]` and `p[1]` (see deal.ii documentation for more information about the class `Point`). In the default cases, the hyperbolic tangent of the signed distance from a circle or sphere is used. In the following example, η will be initialized into a sphere with radius=spanX/8, where $\eta = 0.5$ at the boundary, $\eta \sim 1$ in the interior, and $\eta \sim 0$ outside.

```
template <int dim>
double InitialConditionN<dim>::value (const Point<dim> &p, const unsigned int
    /* component */) const
{
        //set result equal to the structural order paramter initial condition
        double dx=spanX/std::pow(2.0,refineFactor);
        double r=0.0;
        r=p.distance(Point<dim>(spanX/2.0,spanY/2.0,spanZ/2.0));
        return 0.5*(1.0-std::tanh((r-spanX/8.0)/(3*dx)));
}
```

This example requires that `problemDIM`=3. In the actual applications (e.g. `coupledCahnHilliardAllenCahn`), a conditional tests `problemDIM`, and there are analogous cases for each dimension.

3 Boundary Conditions

Boundary conditions are no-flux for the phase field evolution equations. Implementation of alternate boundary conditions is an objective for future releases.

4 Usage

```
$ cmake CMakeLists.txt
$ make
For serial runs:
$ make run
For parallel runs:
$ mpiexec -np nprocs ./main
```